What is ORM?

Object-Relational Mapping

Current Database Code

Manual SQL in PHP:

```php
// Your current approach - lots of SQL!
function get_all_students() {
    $mysqli = new mysqli("localhost", "root", "password", "student_db");
    $result = $mysqli->query("SELECT * FROM students");
    $students = [];
    while ($row = $result->fetch_assoc()) {
        $students[] = $row;
    }
    return $students;
}

function create_student($name, $email, $major, $year) {
    $mysqli = new mysqli("localhost", "root", "password", "student_db");
    $stmt = $mysqli->prepare("INSERT INTO students (name, email, major, year) VALUES (?, ?, ?, ?)");
    $stmt->bind_param("sssi", $name, $email, $major, $year);
    return $stmt->execute();
}
```

50+ lines for basic CRUD operations!

What is ORM?

ORM = **O**bject-**R**elational **M**apping

> ORM translates between **database tables** and **programming objects**.

Simple Analogy:

- **Database Table** = Excel spreadsheet
- **ORM** = Smart translator
- **PHP Object** = Easy-to-use data structure

You work with objects, ORM handles SQL!

Database Table vs PHP Object

Database Table (students):

```
+----+---------+-------------------+--------+-------+
| id | name    | email             | major  | year  |
+----+---------+-------------------+--------+-------+
| 1  | John    | john@example.com  | ASE    | 2     |
| 2  | Sarah   | sarah@email.com   | CS     | 3     |
+----+---------+-------------------+--------+-------+
```

PHP ORM Implementation

We can use PHP to implement ORM.

```php
<?php
class Student {
    public $name;
    public $email;
    public $major;
    public $year;
    private $db;

    public function __construct() {
        // Database connection (simplified)
        $this->db = new PDO('mysql:host=localhost;dbname=university', $username, $password);
    }

    public function save() {
        $sql = "INSERT INTO students (name, email, major, year) VALUES (?, ?, ?, ?)";
        $stmt = $this->db->prepare($sql);
        return $stmt->execute([$this->name, $this->email, $this->major, $this->year]);
    }
}
```

PHP Object Usage (with ORM):

```php
$student = new Student();
$student->name = "John";
$student->email = "john@example.com";
$student->major = "CS";
$student->year = 2;
$student->save();  // ORM converts to SQL automatically!
```

Laravel ORM

However, we can use Laravel ORM that takes care of all the ORM logic behind the scene.

```php
<?php
class Student extends Illuminate\Database\Eloquent\Model
{
    protected $fillable = ['name', 'email', 'major', 'year'];
}
```

```php
<?php
// 2. Usage - Much cleaner!
$student = new Student();
$student->name = "John";
$student->email = "john@example.com";
$student->major = "CS";
$student->year = 2;
$student->save();  // Automatically handles SQL, timestamps, etc.
```

Or even simpler with mass assignment:

```php
Student::create([
    'name' => 'John',
    'email' => 'john@example.com',
    'major' => 'CS',
    'year' => 2
]);
```

Manual SQL vs ORM

Manual SQL (Module 1 Way):

```
// Get all students
$mysqli = new mysqli("localhost", "root", "password", "db");
$result = $mysqli->query("SELECT * FROM students");

// Create student
$stmt = $mysqli->prepare("INSERT INTO students (name, email) VALUES (?, ?)");
$stmt->bind_param("ss", $name, $email);

// Update student
$stmt = $mysqli->prepare("UPDATE students SET name = ? WHERE id = ?");
$stmt->bind_param("si", $name, $id);

// Delete student
$stmt = $mysqli->prepare("DELETE FROM students WHERE id = ?");
$stmt->bind_param("i", $id);
```

ORM (Laravel Eloquent):

```php
// Get all students
$students = Student::all();

// Create student
Student::create(['name' => 'John', 'email' => 'john@email.com']);

// Update student whose id is 1
$student = Student::find(1);
$student->update(['name' => 'Johnny']);

// Delete student
Student::find(1)->delete();
```

4 lines vs 20+ lines with better features!

Why ORM is Amazing

1. No SQL Required

```
// Instead of writing SQL:
"SELECT * FROM students WHERE year = 2 AND major = 'CS'"

// Write readable PHP:
Student::where('year', 2)->where('major', 'CS')->get()
```

2. Automatic Security
    - **Manual SQL**: Risk of SQL injection
    - **ORM**: Automatic SQL injection protection
3. Less Code
    - **Manual**: 200+ lines for CRUD
    - **ORM**: 20 lines for the same functionality

ORM Benefits

Type Safety

```
// Manual SQL - easy to make mistakes
$year = "invalid";  // Oops! Should be integer
$sql = "INSERT INTO students (year) VALUES ($year)";  // SQL error!

// ORM - catches errors early
$student = new Student();
$student->year = "invalid";  // Laravel validation catches this!
```

Relationships Made Easy

```
// Manual SQL - complex joins
"SELECT s.*, c.name as course_name FROM students s
 LEFT JOIN courses c ON s.course_id = c.id WHERE s.id = 1"

// ORM - simple method calls
$student = Student::with('courses')->find(1);
echo $student->courses->name;  // Easy!
```

- From students table, find student with ID = 1
- Also, from the relationship, get all courses that student is enrolled in

```
[
    'id' => 1,
    'name' => 'John', 'email' => 'john@example.com', 'major' => 'CS', 'year' => 2,
    'courses' => [  ['id' => 101, ...], ['id' => 102, ...]]
]
```

Real-World Example

PHP Manual Approach:

```php
function getStudentWithCourses($studentId) {
    $mysqli = new mysqli("localhost", "root", "password", "db");
    // Get student
    $stmt = $mysqli->prepare("SELECT * FROM students WHERE id = ?");
    $stmt->bind_param("i", $studentId);
    $stmt->execute();
    $student = $stmt->get_result()->fetch_assoc();
    // Get courses
    $stmt = $mysqli->prepare("SELECT c.* FROM courses c
                             JOIN student_courses sc ON c.id = sc.course_id WHERE sc.student_id = ?");
    $stmt->bind_param("i", $studentId);
    $stmt->execute();
    $courses = $stmt->get_result()->fetch_all(MYSQLI_ASSOC);

    $student['courses'] = $courses;
    return $student;
}
```

Laravel ORM Approach:

Get student with ID $studentId and load their related courses (eager loading).

```
function getStudentWithCourses($studentId) {
    return Student::with('courses')->find($studentId);
}
```

1 line vs 20+ lines!

The Magic Behind ORM

The Layer of Abstraction on top of raw SQL

The ORM (Object-Relational Mapper) framework provides a layer of abstraction on top of raw SQL.

Class ⇄ Table Mapping

```
use Illuminate\Database\Eloquent\Model;

class Student extends Model {
  protected $table = 'students';   // table name (optional if pluralization fits)
  protected $fillable = ['name','email','major','year']; // mass-assignable cols
}
```

Static methods (all, create, find)

Static Methods = Table-level operations: They don't act on a specific row, but on the whole table.

- Student::all() → ORM runs `SELECT * FROM students`;
- Student::find(1) → ORM runs `SELECT * FROM students WHERE id = 1 LIMIT 1`;
- Student::create([...]) → ORM runs an `INSERT INTO students (...) VALUES (...);`

Instance methods (update, save, delete)

Instance Methods = Row-level operations: Once you already have a model instance (representing a single row), you use instance methods.

- $student->save() → ORM checks if $student already has an id.
  - If no id → it does an `INSERT`.
  - If id exists → it does an `UPDATE`.
- $student->update([...]) → ORM does `UPDATE students SET ... WHERE id = ?`.
- $student->delete() → ORM does `DELETE FROM students WHERE id = ?`.

Function call = SQL statement (via ORM abstraction)

```
// Get all students
$students = Student::all();
// SELECT * FROM students;

// Create student
Student::create(['name'=>'John','email'=>'john@email.com']);
// INSERT INTO students (name,email) VALUES (?, ?);

// Find + Update
$student = Student::find(1);
// SELECT * FROM students WHERE id = 1 LIMIT 1;
$student->update(['name' => 'Johnny']);
// UPDATE students SET name = ? WHERE id = ?;

// Delete
Student::find(1)->delete();
// DELETE FROM students WHERE id = ?;
```

Common ORM CRUD Operations

Create

```
// Manual SQL
$stmt = $pdo->prepare("INSERT INTO students (name, email, major, year) VALUES (?, ?, ?, ?)");
$stmt->execute([$name, $email, $major, $year]);

// Eloquent ORM
Student::create([
    'name' => $name,
    'email' => $email,
    'major' => $major,
    'year' => $year
]);
```

Read

```
// Manual SQL
$stmt = $pdo->query("SELECT * FROM students WHERE major = 'CS'");
$students = $stmt->fetchAll(PDO::FETCH_ASSOC);

// Eloquent ORM
$students = Student::where('major', 'CS')->get();
```

Update

```php
// Manual SQL
$stmt = $pdo->prepare("UPDATE students SET name = ? WHERE id = ?");
$stmt->execute([$newName, $id]);

// Eloquent ORM
Student::find($id)->update(['name' => $newName]);
```

Delete

```php
// Manual SQL
$stmt = $pdo->prepare("DELETE FROM students WHERE id = ?");
$stmt->execute([$id]);

// Eloquent ORM
Student::find($id)->delete();
```

ORM vs Raw SQL: When to Use Each

Use ORM When:

- **95% of applications** - Standard business logic
- **CRUD operations** - Create, read, update, delete
- **Rapid development** - Need to build features quickly
- **Team projects** - Multiple developers working together

Use Raw SQL When:

- **Complex analytics** - Heavy reporting queries
- **Performance critical** - Microsecond optimization needed
- **Legacy systems** - Existing stored procedures

Start with ORM, optimize with SQL only when needed!

Key Takeaways

- ✅ **ORM eliminates SQL complexity** - Focus on business logic
- ✅ **Massive code reduction** - 90% less database code
- ✅ **Better security** - Automatic SQL injection protection
- ✅ **Industry standard** - Expected knowledge for developers